Comprehensive Kernel Instrumentation via Dynamic Binary Translation

Peter Feiner

Angela Demke Brown  Ashvin Goel

University of Toronto
Complexity of Operating Systems
Complexity of Operating Systems

Growth in code size
- Palix, ASPLOS 2011
- Many new drivers!
Complexity of Operating Systems

Growth in code size

- Palix, ASPLOS 2011
- Many new drivers!

More swearing

- Vidar Holen, 2012
Complexity of Operating Systems

Growth in code size
- Palix, ASPLOS 2011
- Many new drivers!

More swearing
- Vidar Holen, 2012

Bugs are inevitable!
- Need coping strategies
Tools would be nice

Awesome tools for **user** code

- Memcheck
- Program Shepherding

Use Dynamic Binary Translation (DBT)

- Rewrite binaries as they execute
- No need for source

Frameworks make building DBT tools easy

- DynamoRIO, Valgrind, Pin

No framework for **OS** code
Our Framework
Our Framework

Ported DynamRIO to Linux kernel

- Runs on bare metal
Our Framework

Ported DynamRIO to Linux kernel

- Runs on bare metal

Port took 18 Months
Our Framework

Ported DynamRIO to Linux kernel

- Runs on bare metal

Port took **18 Months**

Built OS debugging tools in **5 days**

- Heap debugging
  - Use after free
  - Heap corruption
- Stack overflow monitor
Our Framework

Ported DynamRIO to Linux kernel

- Runs on bare metal

Port took 18 Months

Built OS debugging tools in 5 days

- Heap debugging
  - Use after free
  - Heap corruption
- Stack overflow monitor

Practical

- One author ran system on her desktop for 1 month
What About Hypervisors?
What About Hypervisors?

VMWare can use DBT on guests

- No instrumentation API
What About Hypervisors?

VMWare can use DBT on guests

- No instrumentation API

PinOS has instrumentation API

- PinOS = Pin + Xen
- Guest needs emulated devices
  - Useless for most driver code
What About Hypervisors?

VMWare can use DBT on guests
  - No instrumentation API

PinOS has instrumentation API
  - PinOS = Pin + Xen
  - Guest needs emulated devices
    - Useless for most **driver code**

Palix, ASPLOS 2011
What About Hypervisors?

VMWare can use DBT on guests
  - No instrumentation API

PinOS has instrumentation API
  - PinOS = Pin + Xen
  - Guest needs emulated devices
    - Useless for most driver code

So add DBT to a hypervisor with pass-through devices?
  - Then you’d have the problems we show you how to solve
  - ... problems with interrupts!
Framework Overview

1. Boot normally

2. Take over

3. JIT the OS

\[ \text{x86} \rightarrow \text{instrumented x86} \]
Framework Overview

1. Boot normally

2. Take over

3. JIT the OS
   x86 → instrumented x86

Diagram:
- OS Code
- Framework
- User Code
- Interrupts
- Exceptions
Framework Overview

1. Boot normally

2. Take over

3. JIT the OS

x86 → instrumented x86
Dynamic Instrumentation

- User Code
- Framework
- Instrumented OS Code
- OS Code
- Interrupts
- Exceptions
- Exceptions
- Exceptions
Dynamic Instrumentation

- OS Code
- Framework
  - User Code
    - Code Cache
  - Interrupts
  - Exceptions
Dynamic Instrumentation

OS Code

Code Cache

User Code

Framework

bb1

bb2

bb3

Interrupts

Exceptions
Dynamic Instrumentation

OS Code

<table>
<thead>
<tr>
<th>bb1</th>
</tr>
</thead>
<tbody>
<tr>
<td>bb2</td>
</tr>
<tr>
<td>bb3</td>
</tr>
</tbody>
</table>

Code Cache

User Code

Framework
Dynamic Instrumentation

System call example

‣ bb1 is entry point
Dynamic Instrumentation

OS Code

Code Cache

User Code

Framework

System call example

- bb1 is entry point
Dynamic Instrumentation

System call example

- bb1 is entry point
Dynamic Instrumentation

System call example

- bb1 is entry point
Dynamic Instrumentation

System call example

- bb1 is entry point
System call example

- bb1 is entry point
Dynamic Instrumentation

System call example
- bb1 is entry point
Dynamic Instrumentation

System call example

- bb1 is entry point
System call example

- bb1 is entry point
Dynamic Instrumentation

System call example

- bb1 is entry point
Dynamic Instrumentation

OS Code

bb1

bb2

bb3

Code Cache

bb1

bb2

User Code

Framework

System call example

- bb1 is entry point
Dynamic Instrumentation

System call example
- bb1 is entry point
Dynamic Instrumentation

OS Code

User Code

Code Cache

System call example

- bb1 is entry point
Dynamic Instrumentation

System call example

- bb1 is entry point
Dynamic Instrumentation

System call example
- bb1 is entry point
Complications

OS Code

bb1

bb2

bb3

Code Cache

bb1

bb2

bb3

User Code

Framework
Complications

Reentrance
- How do you do I/O?
- Can’t use OS
Complications

OS Code

Reentrance
- How do you do I/O?
- Can’t use OS

Code Cache

Concurrency
- Multiple CPUs using and building cache
Complications

Reentrance
- How do you do I/O?
- Can’t use OS

Interrupts

Concurrency
- Multiple CPUs using and building cache
Handling Interrupts

OS Code

User Code

Framework

interrupts

OS Code

User Code

Framework

interrupts

Code Cache
Handling Interrupts

OS Code

bb1

IH

* iret

Framework

User Code

Interrupts

Code Cache

bb1
Handling Interrupts

OS Code

Code Cache

User Code

Framework

Interrupts

arrival

bb1

iret

*
Handling Interrupts

OS Code

bb1

IH

* iret

Code Cache

arrival

bb1

User Code

Framework

Interrupts
Handling Interrupts

What should framework do with interrupt?
What should framework do with interrupt?

- Can it run interrupt handler IH immediately?
Handling Interrupts

What should framework do with interrupt?

- Can it run interrupt handler IH immediately?
Handling Interrupts

What should framework do with interrupt?

- Can it run interrupt handler IH immediately?
What should framework do with interrupt?

- Can it run interrupt handler IH immediately?
What should framework do with interrupt?

- Can it run interrupt handler IH immediately?
Handling Interrupts

What should framework do with interrupt?

- Can it run interrupt handler IH immediately?
  - Problem if *instrumentation* isn’t reentrant
What should framework do with interrupt?

- Can it run interrupt handler IH immediately?
  - Problem if *instrumentation* isn’t reentrant
Handling Interrupts

What should framework do with interrupt?

- Can it run interrupt handler IH immediately?
  - Problem if instrumentation isn’t reentrant
Handling Interrupts

What should framework do with interrupt?

- Can it run interrupt handler IH immediately?
  - Problem if instrumentation isn’t reentrant
Handling Interrupts

What should framework do with interrupt?

- Can it run interrupt handler IH immediately?
  - Problem if instrumentation isn’t reentrant

Need to delay
Delaying Interrupts
Delaying Interrupts

Where do we delay it until?

Framework

bb l

arrival
Delaying Interrupts

Where do we delay it until?

Delay until end of bb1?

Framework

bb1

delivery

arrival
Delaying Interrupts

Where do we delay it until?

Delay until end of bb1?
  • Avoids tail
Delaying Interrupts

Where do we delay it until?

Delay until end of bb1?

- Avoids tail

[Diagram of(bb1, delivery, Framework, pending, interrupt)]
Delaying Interrupts

Where do we delay it until?

Delay until end of bb1?

- Avoids tail
Delaying Interrupts

Where do we delay it until?

Delay until end of bb1?

- Avoids tail
Delaying Interrupts

Where do we delay it until?

Delay until end of bb1?

- Avoids tail

- Framework pending interrupt

- bb1 arrival

- delivery
Delaying Interrupts

Where do we delay it until?

Delay until end of bb1?

- Avoids tail
Delaying Interrupts

Where do we delay it until?

Delay until end of bb1?

- Avoids tail
- Problem if bb1 disables interrupt
Delaying Interrupts

Where do we delay it until?

Delay until end of bb1?

- Avoids tail
- Problem if bb1 disables interrupt
  - Could be any MMIO
  - Framework cannot detect if enabled
Delaying Interrupts

Where do we delay it until?

Delay until end of bb1? ✗

- Avoids tail
- Problem if bb1 disables interrupt
  - Could be any MMIO
  - Framework cannot detect if enabled
Delaying Interrupts

Where do we delay it until?

Delay until end of bb1? ✗

- Avoids tail
- Problem if bb1 disables interrupt
  - Could be any MMIO
  - Framework cannot detect if enabled

Must deliver before next OS instruction
Delaying Interrupts

Where do we delay it until?

Delay until end of bb1? ✗

- Avoids tail
- Problem if bb1 disables interrupt
  - Could be any MMIO
  - Framework cannot detect if enabled

Must deliver before next OS instruction

- Delay until end of instrumentation
Delaying Interrupts

Where do we delay it until?

Delay until end of bb1? ✗

- Avoids tail
- Problem if bb1 disables interrupt
  - Could be any MMIO
  - Framework cannot detect if enabled

Must deliver before next OS instruction

- Delay until end of instrumentation
- Still duplicates tail
How to Delay Interrupts

bb1
How to Delay Interrupts

Could disable them on the CPU
How to Delay Interrupts

Could disable them on the CPU
push, disable

bb l
How to Delay Interrupts

Could disable them on the CPU

push, disable pop
How to Delay Interrupts

Could disable them on the CPU

push, disable  pop

push, disable

bb l
How to Delay Interrupts

Could disable them on the CPU

push, disable  pop

push, disable  pop
How to Delay Interrupts

Could disable them on the CPU

push, disable pop

But performance would be bad
How to Delay Interrupts

Could disable them on the CPU

push, disable pop

But performance would be bad

Instead, have framework handle it

- Extra overhead for interrupt, cheaper instrumentation
How to Delay Interrupts

Could disable them on the CPU

push, disable  pop

But performance would be bad

Instead, have framework handle it

- Extra overhead for interrupt, cheaper instrumentation
- Instrumentation more frequent than interrupts
- Gigabit NIC sends interrupt every 100µs ≈ 100K instr.
Delaying with Patches

Example: interrupt 239
Delaying with Patches

Example: interrupt 239

Interrupt Stack Frame

<table>
<thead>
<tr>
<th>Interrupts enabled</th>
<th>yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Delaying with Patches

Example: interrupt 239

Interrupt Stack Frame

<table>
<thead>
<tr>
<th>Interrupts enabled</th>
<th>yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Example: interrupt 239

The framework

1. Patches next native instruction

Interrupt Stack Frame

<table>
<thead>
<tr>
<th></th>
<th>yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupts enabled</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Delaying with Patches

Example: interrupt 239

The framework

1. **Patches** next native instruction
2. **Disables** interrupts on iret

---

### Interrupt Stack Frame

<table>
<thead>
<tr>
<th>Interrupts enabled</th>
<th>no</th>
<th>yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Diagram:
- **Framework**
- **bb l**
- **arrival**
- **int 239**
Delaying with Patches

Example: interrupt 239

The framework

1. **Patches** next native instruction
2. **Disables** interrupts on iret
3. iret

---

<table>
<thead>
<tr>
<th>Interrupt Stack Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupts enabled</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

...
Delaying with Patches

Example: interrupt 239

The framework

1. **Patches** next native instruction
2. **Disables** interrupts on iret
3. iret

<table>
<thead>
<tr>
<th>Framework</th>
<th>arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bb1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interrupt Stack Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupts enabled</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

| Interrupts enabled    | yes     |

Example: int 239
Delaying with Patches

Example: interrupt 239

The framework

1. **Patches** next native instruction
2. **Disables** interrupts on iret
3. iret

---

## Interrupt Stack Frame

<table>
<thead>
<tr>
<th>Interrupts enabled</th>
<th>no</th>
<th>yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Delaying with Patches

Example: interrupt 239

The framework

1. **Patches** next native instruction
2. **Disables** interrupts on iret
3. iret

---

### Interrupt Stack Frame

<table>
<thead>
<tr>
<th></th>
<th>yes</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupts enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Patch Interrupt Stack Frame

<table>
<thead>
<tr>
<th></th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupts enabled</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Delaying with Patches

Example: interrupt 239

The framework

1. **Patches** next native instruction
2. **Disables** interrupts on iret
3. iret

Interrupt Stack Frame

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupts enabled</td>
<td>yes</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Patch Interrupt Stack Frame

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupts enabled</td>
<td>no</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

```
bb 1
int 239
arrival
```
Delaying with Patches

Example: interrupt 239

The framework

1. **Patches** next native instruction
2. **Disables** interrupts on iret
3. iret
4. Removes patch

Interrupt Stack Frame

<table>
<thead>
<tr>
<th>Interrupts enabled</th>
<th>no</th>
<th>yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Patch Interrupt Stack Frame

<table>
<thead>
<tr>
<th>Interrupts enabled</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
## Delaying with Patches

**Example:** interrupt 239

The framework:

1. **Patches** next native instruction
2. **Disables** interrupts on iret
3. iret
4. Removes patch
5. **Enables** interrupts on iret

### Interrupt Stack Frame

<table>
<thead>
<tr>
<th></th>
<th>Interrupts enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

### Patch Interrupt Stack Frame

<table>
<thead>
<tr>
<th></th>
<th>Interrupts enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>no</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- Label: "arrival" with an arrow pointing to "bb 1" and "int 239"
Delaying with Patches

Example: interrupt 239

The framework

1. **Patches** next native instruction
2. **Disables** interrupts on iret
3. iret
4. Removes patch
5. **Enables** interrupts on iret
6. Run instrumented interrupt handler

<table>
<thead>
<tr>
<th>Interrupt Stack Frame</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupts enabled</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Patch Interrupt Stack Frame</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupts enabled</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Delaying with Patches

Example: interrupt 239

The framework

1. **Patches** next native instruction
2. **Disables** interrupts on iret
3. iret
4. Removes patch
5. **Enables** interrupts on iret
6. Run instrumented interrupt handler

### Interrupt Stack Frame

<table>
<thead>
<tr>
<th>Interrupts enabled</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

### Patch Interrupt Stack Frame

<table>
<thead>
<tr>
<th>Interrupts enabled</th>
<th>yes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Delaying with Patches

Example: interrupt 239

The framework

1. **Patches** next native instruction
2. **Disables** interrupts on iret
3. **iret**
4. Removes patch
5. **Enables** interrupts on iret
6. Run instrumented interrupt handler

**Okay, what’s the performance?**
Performance

Ran framework with instruction counting tool

- Intel Quad Core i7 2.8Ghz, 8GB, 64-bit Ubuntu 10.10

Low application overhead

- JavaScript, Mozilla Kraken: 3% overhead
- Parallel Linux kernel compile: 30% overhead
  - 18% user time increase
  - 143% system time increase

Overhead commensurate with OS activity

- How bad can this get?
Stress Test Setup

Apachebench and Filebench

Configured benchmarks to stress CPUs and kernel

- Large buffer cache - no disk I/O
- Many threads - lots of context switching
- 100% utilization - shows interrupt processing overhead

<table>
<thead>
<tr>
<th></th>
<th>nthreads</th>
<th>data size</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileserver</td>
<td>50</td>
<td>1.25 GB</td>
</tr>
<tr>
<td>webserver</td>
<td>100</td>
<td>15.6 MB</td>
</tr>
<tr>
<td>webproxy</td>
<td>100</td>
<td>15.6 MB</td>
</tr>
<tr>
<td>varmail</td>
<td>16</td>
<td>15.6 MB</td>
</tr>
</tbody>
</table>

Table 1. Filebench parameters

concurrency level 200

Apachebench Parameters
Stress Test Results

Less than 5x - Reasonable overhead for debugging tools
Stress Test Results

Throughput (ops/s)

Apachebench

Filebench

Native
Instrumented

Less than 5x - Reasonable overhead for debugging tools

CPU-bound

IO-bound

apachebench

fileserv

webserver

webproxy

varmail
Summary

Enables dynamic binary instrumentation of OS

 Makes it easy to write complex instrumentation

Built useful memory checking tools

Works with arbitrary devices & drivers
Questions?